Intelligent Force Sensor

レシーバーボード

取扱説明書

ニッタ株式会社
ＲＥＴＳ事業部
センサーグループ

5907Cj1.4

マニュアルの概観 1
マニュアルのレイアウト 1
まず最初に 1
アーキテクチャーの概観 2
ＤＳＰベース力センサレシーバのデータロケーションおよび定義 5
FORCE_SENSOR_DATA 5
RAW_CHANNELS 5
COPYRIGHT 5
SHUNTS 6
DEFAULT_FS 6
LOAD_ENVELOPE_NUM 6
MIN_FULL_SCALE 6
TRANSFORM_NUM 7
MAX_FULL_SCALE 7
PEAK_ADDRESS 7
FULL_SCALE 7
OFFSETS 8
OFFSET_NUM 8
VECT_AXES 8
FILTER0 8
FILTER1_FILTER6 9
RATE_DATA 9
MINIMUM_DATA 9
MAXIMUM_DATA 9
NEAR_SAT_VALUE 10
SAT_VALUE 10
RATE_ADDRESS 10
RATE_DIVISOR 10
RATE_COUNT 10
COMMAND_WORD2 11
COMMAND_WORD1 11
COMMAND_WORD0 11
COUNT1_COUNT6 11
ERROR_COUNT 11
COUNT_X 12
WARNINGS 12
ERRORS 12
THRESHOLD_BITS 12
LAST_CRC 12
EEPROM_VER_NO 12
SOFTWARE_VER_NO 12
SOFTWARE_DAY 13
SOFTWARE_YEAR 13
SERIAL_NO 13
MODEL_NO 13
CAL_DAY 13
CAL_YEAR 13
UNITS 14
BITS 14

CHANNELS....14
THICKNESS....14
LOAD_ENVELOPES....14
TRANSFORMS....15
データストラクチャの詳細....16
RAW_CHANNEL....16
RAW_TIME....16
RAW_DATA....16
FORCE_ARRAY....17
SIX_AXIS_ARRAY....17
VECT_BITS....18
WARNING_BITS....18
XX_NEAR_SAT....18
ERROR_BITS....19
XX_SAT....19
MEMORY_ERROR....19
SENSOR_CHANGE....19
SYSTEM_BUSY....19
CAL_CRC_BAD....19
WATCH_DOG....20
WATCH_DOG2....20
FORCE_UNITS....20
THRESH_STRUCT....21
DATA_ADDRESS....21
THRESHOLD....21
BIT_PATTERN....21
LE_STRUCT....21
LATCH_BITS....21
NUMBER_OF_GE_THRESHOLDS....22
NUMBER_OF_LE_THRESHOLDS....22
LINK_TYPES....23
TRANSFORM....24
Ｃ言語入門....25
コマンドの詳細....27
コマンド例（０）....27
メモリリード（１）....28
メモリライト（２）....28
ビットセット（３）....29
ビットリセット（４）....30
座標変換番号（５）....31
オフセット番号（６）....32
セットオフセット（７）....33
リセットオフセット（８）....34
セットベクトル（９）....35
セットフルスケール（１０）....36
リセットピーク（１１）....37
リードピーク（１２）....38
コマンドの使用例....39
例１－ソフトウェアバージョンの表示....39
例２－計算後のＦｘ力データの表示....39

例３－オフセットのリセット 39
例４－Ｆｚのオフセット変更 40
例５－ベクトルの設定 40
例６－座標変換 41
例７－複雑な座標変換 41
例８－ロードエンベロープ 42
データロケーション 43
コマンド一覧 44
用語の説明 45
処理速度について 46
ＩＳＡ－２１０５ 48

マニュアルの概観

本マニュアルは、レシーバボードのセットアップ及び操作法を説明するものです。本マニュアルの大部分は、各種レシーバボードに共通な情報を説明します。個別のレシーバボードに対する情報は、付録において説明します。
本概観は ３つの項から成り立っています。最初にマニュアルのレイアウトを、次に結果を迅速に得ていただくための、いくつかの提案をする”スタートの前に”を示します。最後に、レシーバボードのアーキテクチャーの概観を示します。

マニュアルのレイアウト

レシーバボードは、ユーザー及びボード自体がともに読み書きできるデュアルポートＲＡＭを持っています。ボードは力モーメントデータをこのＲＡＭに書き込みます。それからユーザーは、このデータをＲＡＭから読み込むことができます。本マニュアルでは、主にこの共有領域を説明することになります。データ構造の宣言は、Ｃ言語スタイルコードでフォーマットされます。第１項は「データロケーション及び定義」で 5 ページから始まり、変数の宣言及び記述について説明します。第２項「データストラクチャの定義」は 16 ページから始まり、データ型宣言をします。このふたつの項のデータ宣言はＣ言語スタイルで示されます。簡単な「Ｃ言語入門」を 25 ページに示しました。
次の項「コマンドの定義」は 27 ページから始まり、ボードが受け付け可能なコマンドを説明しています。これらのコマンドは、いろいろなデータ位置とともに、レシーバボードによってなされるタスクを変更するために使われます。39 ページからは、使用例を説明します。その後は、データ位置及びコマンドをまとめた表、用語の説明、性能に関する簡単な考案と続きます。付録にはレシーバの各々特定バージョンのデータを示します。

まず最初に

本マニュアルでまず最初に読んでいただきたいのは、本項に続く「アーキテクチャーの概観」の項です。そこでは、レシーバボードの能力について考案しています。次に目を通していただきたいのは、御購入いただいたレシーバに関し、説明している付録の部分です。そこには、ハードウェア及びソフトウェアからレシーバをどのようにインタフェースするかが示されています。
もしも、Ｃ言語に関してあまり知識や経験のない場合には、25 ページの「Ｃ言語入門」を読んで下さい。Ｃ言語の簡単な説明を読んだあとに、「データロケーション及び定義」の項、及び「データストラクチャの定義」の項にざっと目を通して下さい。最後に、「コマンドの定義」の項及びコマンドの実行法を説明した「使用例」を読んで下さい。
ボードとはじめて交信しようとするときには、アドレス 0x0040 の著作権表示(copyright statement) 及びアドレス 0x00ef の count_x 変数を読み込んで下さい。共有アドレス空間からデータを読むことができるかどうかがすぐにわかります。また、0x0088 の Fx オフセット変数にある値を書き込み、それからその変数を読み込んで下さい。そうすると 共有アドレス空間への書き込みが成功したかどうかがすぐに判ります。

アーキテクチャーの概観

レシーバボードには、６自由度(6DOF)の力及びモーメントデータを極めて広い帯域幅で取得可能とする最新技術が用いてられています。10 MipsDSP チップであるアナログデバイセズ社の ADSP-2105 を使用することによって、本システムは干渉除去され、デジタルフィルタリングされたデータを１チャネルあたり 8KHz で取得可能になりました。このデータ取得速度は、従来品に比べ１０倍もの高速となりました。レシーバは内蔵電子回路を有する本センサと協調して動作するように最適化されています。

レシーバボードには、ＩＢＭ－ＡＴバス、ＶＭＥバス、ＮＥＣ－９８バスを含む数種のバスに対応したバージョンがあります。これらのボードは共通のアーキテクチャーを持っています。レシーバボードはデュアルポートＲＡＭを有し、ホスト及びボードの両方から読み書きができます。このＲＡＭは、ホストがボードからデータを、ほとんどオーバーヘッドなしに読み込むことを可能にします。また、ホストがＲＡＭにコマンドを書き込むことにより動作中であってもボードの設定を変更することが出来ます。

レシーバボードには、センサからのシリアルデジタル転送データを受信する回路の他にセンサへの供給電圧をモニタし調節する回路が搭載されています。自動的に遠隔供給電圧調整ができるということは、センサケーブルに対する要求が極めてゆるくなるということを意味しています。長くて、細いゲージワイヤも使用可能となります。したがって、この内蔵電子回路を有する最新のセンサでは、もはや硬くて高価なケーブルは必要ありません。

デュアルポートアドレス空間は、１６ｋの２バイトワードで構成されます。この空間の最初の８ｋはＲＡＭで、残りの８ｋはスタータスレジスタや他の機能を構成します。たいていの仕事は、共有アドレス空間の最初の２５６ワードで行われます。共有アドレス空間の変数の位置については、本マニュアル 5 ページの「データ位置及び定義」９項に説明してあります。共有アドレス空間への読み書きの詳細は、レシーバによってちがうので、付録に各々のレシーバについて別個に説明してあります。

ＤＳＰはセンサから送られてきた生データを処理するために使われます。ＤＳＰは、次のような処理をします。オフセットの除去、干渉除去、飽和の検出、デジタルローパスフィルタリング、力及びモーメントベクトルの大きさの計算、ピークの検出、レイトの計算、座標変換 及び 閾値のモニタリングなどです。

センサから出力された生データは、干渉除去マトリクスを通過し、オフセット除去されます。このプロセスにより風袋重量の除去とともに、センサのクロストークの除去がなされます。このプロセスを経由したデータでは、A/D 変換器が飽和しているかどうかの判断が難しくなることがあります。A/D 変換器が飽和すると、力データを用いたフィードバックループは不安定になります。それを防止するために、ＤＳＰはセンサの生データをモニタし、飽和が近づいたときや飽和したときにアラームを発生します。

干渉除去データは、カスケードローパスフィルタを通ります。各フィルタは４データ毎に計算され、そのカットオフ周波数は前のフィルタの 1/4 のカットオフ周波数となります。したがって サンプルレイトが 8KHz の標準センサでは、最初のフィルタのカットオフ周波数は 500Hz となります。さらに続くフィルタのカットオフ周波数は、125Hz, 31.25Hz, 7.81Hz ---- となります。フィルタの遅延時間は、次式で近似されます。

遅延時間 ＝ １/ｶｯﾄｵﾌ周波数

したがって 500Hz のフィルタの遅延時間は、

１/ 500Hz ＝ ２ ms

となります。しかし、遅延時間はデータの周波数によって変化するので、この値は 良好な近似値とはなりますが、正しい値ではありません。正確なフィルタ特性については、フィルタ応答を示した添付図を参照して下さい。

力モーメントベクトルの大きさは各々のデータ成分から計算されます。したがって、ベクトルにはフィルタリングされていない干渉除去データから計算されるものと、フィルタリングされたデータから計算されたものがあります。フィルタリングされていないベクトルは、フィルタリングされていないデータの２分の１の帯域幅で計算されます。フィルタリングされたデータから計算されるベクトルは、フィルタのカットオフ周波数の４倍で計算され、フィルタリング計算の１/４の頻度で計算されます。

どの単一のデータセット（８項目）でもピーク（極大）と谷（極小）をモニタすることができます。データは全帯域幅でモニタされ、新しく最大値、最小値が検出されるとストアされます。その最大値、最小値はユーザコントロール下で読み込みやリセットができます。ユーザ設定間隔で、データセットの１次導関数も計算され、出力されます。

センサ軸の原点及び方向を変更するために、座標変換を行うことができます。この機能を使うと、たとえばセンサ座標をツール座標に合わせることができるため、力制御に必要とされる数学演算を非常に単純化することができます。

閾値もモニタできます。ＤＳＰにはユーザ設定閾値をデータがクロスしたときに、あるビットをトグルされる機能があります。ロードエンベロープ機能を使用することによって、非常に小さいオーバーヘッドで複数個の閾値制御を行うことが出来ます。

次のグラフはデジタルフィルタの特性を示します。このグラフでは、最初の３つのフィルタの応答を示しました。フィルタの出力は、そのカスケード特性のために、フィルタ周波数が低下するにつれてわずかに変化します。
３番目以降のフィルタデータは、本質的にフィルタ#3 と同じなので、ここでは省略しました。フィルタ#4～#6 でただひとつちがうのは周波数の軸を順次フィルタ#3 のグラフを出発点として４分の１する必要があるということです。

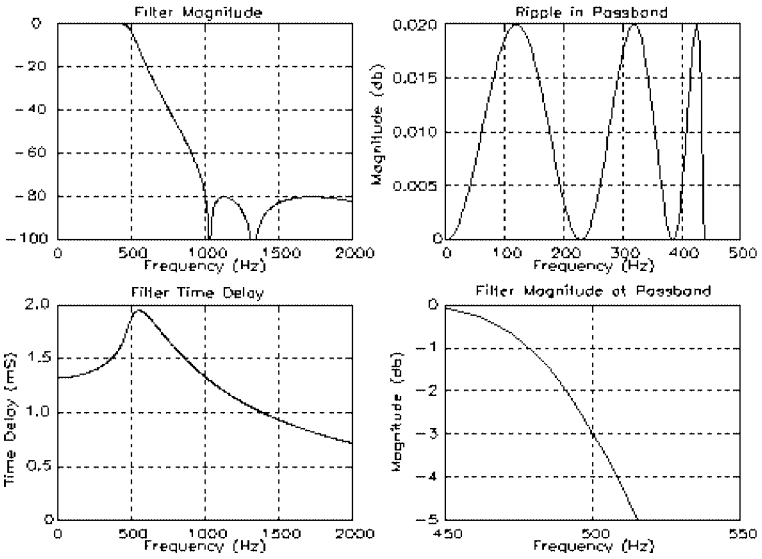

図１
フィルタ１の特性図

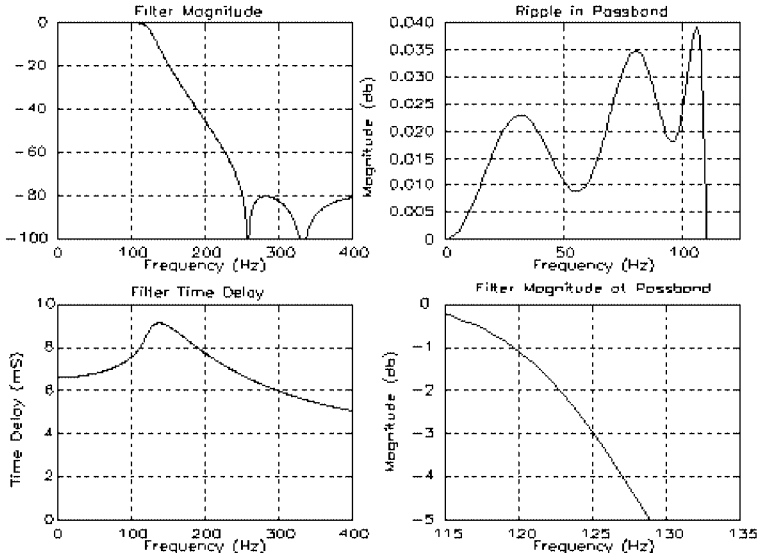

図２
フィルタ２の特性図

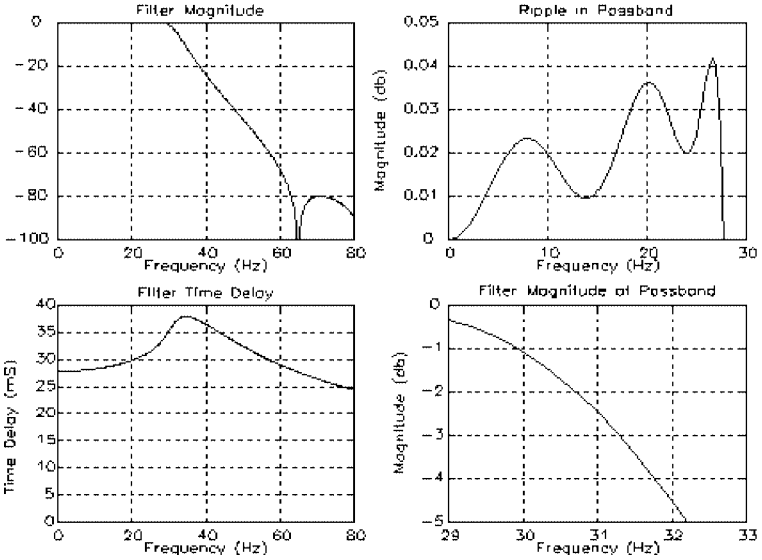

図３
フィルタ３の特性図

ＤＳＰベース力センサレシーバのデータロケーションおよび定義

次の情報は、ＤＳＰベース力センサレシーバのメモリインタフェースの詳細を示します。データストラクチャー宣言は、”Ｃ言語”形式で表示しています。サポートするストラクチャー宣言は、データストラクチャー宣言に続きます。宣言はクーリエ型文字でタイプされ、他の記述と区別しやすいようにしました。reserved と書かれた領域は、ユーザーによって書き込むことはできません。43 ページに変数の図表を、45 ページに用語の解説集を示しました。

```
/*
     For the following structure definitions :

     int:       signed 16 bit value
     unsigned int:   unsigned 16-bit value

     bit fields are shown with the lsb first

*/
```

FORCE_SENSOR_DATA

force_sensor_data ストラクチャーは、レシーバーボード上のデータのレイアウトを示します。示されたオフセットはＤＳＰアドレス空間へのオフセットで、１６ビット長のデータ空間用です。したがって、メモリが一度に８ビットしかアドレスされないような環境では、オフセットは２倍する必要があります。

```
struct force_sensor_data
{
```

RAW_CHANNELS

センサから出力される生データ(raw data)をストアするための領域です。

```
     raw_channel[16]      raw_channels         /* offset 0x0000 */
```

COPYRIGHT

著作権の公示を含んだ null terminated ASCII 文字列です。

```
     int[0x0018]          copyright;           /* offset 0x0040 */
     int[0x0008]          reserved1;           /* offset 0x0058 */
```

SHUNTS

センサのシャントデータが書き込まれています。回路外付型センサにはそのゲインを調整する機能があります。この機能によって用途に適した分解能やダイナミックレンジが得られるようにハードウェアのフルスケールを調整することができます。本機能を有するセンサでは、キャリブレーション時に各々のセンサチャンネルのゲインを、シャント抵抗を使って測定することが出来ます。シャント抵抗は、抵抗ブリッジのひとつの抵抗に並列に結合され、そのために生じる出力変化が測定されます。このデータをシャントデータと呼び、ここに記録されます。センサのゲインを変更しようとするときには、ここでシャント測定を行うことができます。ＤＳＰはキャリブレーションマトリックスにスケールをかけて、指示されたシャントの読みに適合したゲインを求めます。もしも、シャントが０であると、センサはそのゲインを変更することができません。センサゲインの変更、シャントデータの詳細については、回路外付型センサマニュアルを参照してください。これらの値を発効させるためには、コマンド(5) use tranform# (p.31) またはコマンド(10) set new full scales (p.36) をコールしなければなりません。

```
six_axis_array      shunts;             /* offset 0x0060 */
int[2]              reserved2;          /* offset 0x0066 */
```

DEFAULT_FS

フルスケールを指定していない場合に用いられるフルスケールが書き込まれています。

```
six_axis_array      default_FS;         /* offset 0x0068 */
int                 reserved3;          /* offset 0x006e */
```

LOAD_ENVELOPE_NUM

現在使用されているロードエンベロープの番号です。この値は、ロードエンベロープの設定を行った後にセットして下さい。

```
int                 load_envelope_num;  /* offset 0x006f */
```

MIN_FULL_SCALE

推奨する最小のフルスケールです。
この値は max_full_scale(p.7)とともに、フルスケールを設定するための適切な値を決定するのに役立ちます。ソフトウェアによりセンサのフルスケールを任意の値に設定することができます。しかし、フルスケールを適切に設定しないと不都合が起こる場合があります。フルスケールをあまりに低く設定すると、データはすぐに飽和してしまうし、ダイナミックレンジも失われてしまいます。また、フルスケールを大きく設定しすぎると、データが右にシフトするにしたがって分解能もなくなり、lsb も失われてしまいます。したがって、最大フルスケールは分解能が失われない最大の値であり、最小フルスケールはデータがすぐに飽和してしまわない値としなければなりません。これらの値は、新しい座標変換が計算されるときには、いつでも計算されます。推奨最大値が 推奨最小値よりも小さいということがあり得ます。これは、主に座標変換を使用したときにおこります。このような場合には、フルスケー

ルの選択は ダイナミックレンジと分解能の間での妥協点ということになります。通常は分解能を優先して、推奨最大フルスケールを選択することをお勧めします。

警告：フルスケールが、推奨最小フルスケールの 0.4%よりも大きいこと確認してください。この値よりもフルスケールが小さいと誤った結果が出ます。

```
six_axis_array      min_full_scale;     /* offset 0x0070 */
int                 reserved4;          /* offset 0x0076 */
```

TRANSFORM_NUM

現在使用中の座標変換番号です。この値は、コマンド(5)use transform # (p.31) 実行後に、ＤＳＰによってセットされます。

```
int                 transform_num;      /* offset 0x0077 */
```

MAX_FULL_SCALE

推奨する最大のフルスケールです。詳細については min_full_scale (p.6) の項を参照して下さい。

```
six_axis_array      max_full_scale;     /* offset 0x0078 */
int                 reserved5;          /* offset 0x007e */
```

PEAK_ADDRESS

ここにはピークルーチンでモニタするデータのアドレスを書き込みます。この値はユーザにより設定されます。ピークルーチンはピーク値を示す８つの連続するアドレスをモニタします。（例：フィルタ３のピーク値を見るためには、この値を 0x00a8 に設定して下さい。）

```
int                 peak_address;       /* offset 0x007f */
```

FULL_SCALE

現在使用中のセンサフルスケールを示します。干渉除去されフィルタリングされたデータは処理されているので、±16384 がフルスケールに等しくなります。使用する工学単位については、14 ページに示します。Fx, Fy, Fz, Mx, My 及び Mz のフルスケールは、コマンド(10) set new full scale (p.36) を呼び出す前に書き込むことができます。V1, V2 のフルスケールはある軸のフルスケールが変更されるか、またはベクトルを計算に使用する軸が変更されるときに設定されます。V1, V2 のフルスケールは常に、各々のベクトルに使用される軸のうちで最大のフルスケールに等しくなります。

```
force_array         full_scale;         /* offset 0x0080 */
```

OFFSETS

センサのオフセットが書き込まれています。これらの値は干渉除去データを得るためにセンサデータから引かれます。オフセットは キャリブレーションデータが受け取られてから数秒間（＜１０）でセットされます。オフセットは出力データがゼロになるように設定されます。これらの値は書き込みも読み込みもできます。ＤＳＰは 書き込まれた値を、書き込み後 2ms 以内で使用することが出来ます。干渉除去データをゼロにするためには、この値を現在の干渉除去データに加えて、その和を設定するようにして下さい。ＤＳＰは、新しい座標変換が適用されたときにこれらの値を変更します。オフセットが Fx は５で、他のすべてがゼロに設定されていると、Ｚ軸についての９０°回転後には、Fy が５で他のすべてはゼロになります。

```
six_axis_array      offsets;            /* offset 0x0088 */
```

OFFSET_NUM

現在使用中のオフセット番号を示します。この値は、use offset # コマンド(p.32) が実行されて後、ＤＳＰによって設定されます。その値は 0 ～ 15 です。

```
int                 offset_num;         /* offset 0x008e */
```

VECT_AXES

ベクトル計算にどの軸が使われているかを示すビットマップです。set vector axes コマンド(p.35)が実行された後にＤＳＰにより設定されます。

```
vect_bits           vect_axes;          /* offset 0x008f */
```

FILTER0

干渉除去されているがフィルタリングされていないデータを示します。この値はオフセット除去されています。

```
force_array         filter0;            /* offset 0x0090 */
```

FILTER1_FILTER6

フィルタリングされたデータです。干渉除去されたデータはカスケードローパスフィルタを通過します。filter 1 **のカットオフ周波数はセンサのサンプルレイトの** 1/16 **です。**8KHz **のサンプルレイトのセンサでは、**filter1 **のカットオフ周波数は** 500Hz **になります。以後のフィルタでは、カットオフ周波数は** 125Hz, 31.25Hz, 7.813Hz, 1.953Hz, 0.4833Hz **となります。**

```
force_array          filter1;             /* offset 0x0098 */
force_array          filter2;             /* offset 0x00a0 */
force_array          filter3;             /* offset 0x00a8 */
force_array          filter4;             /* offset 0x00b0 */
force_array          filter5;             /* offset 0x00b8 */
force_array          filter6;             /* offset 0x00c0 */
```

RATE_DATA

計算されたレイトデータです。これは１次導関数です。変数 rate_divisor**(**p.10**) により指定された周期で計算されます。レイトが計算されるデータは 変数** rate_address **に示されます。**

```
force_array          rate_data;           /* offset 0x00c8 */
```

MINIMUM_DATA
MAXIMUM_DATA

最小データ値、最大（ピーク）データ値です。ＤＳＰはセンサの全帯域で 最小値及び最大値を示す８つの連続するデータをモニタできます。この領域は、ユーザリクエストによってのみ更新されます。いかなるピークも見逃さないように実行されます。データを読むためには、read peaks **コマンド(**p.38**) 又は** read and reset peaks **コマンド(**p.37**) を使って下さい。ピークデータは座標変換やフルスケール変更の実行時に消失します。また、新しいセンサをプラグインしたときにも失われます。**

```
force_array          minimum_data;        /* offset 0x00d0 */
force_array          maximum_data;        /* offset 0x00d8 */
```

NEAR_SAT_VALUE
SAT_VALUE

生データの飽和を判断する基準値が書き込まれています。干渉除去やオフセット除去後の処理データからセンサが飽和しているかどうかを判断することは困難なことです。これらの値は error and warning words (p.12) とともに、このような重要な情報を与えてくれます。これらの２つの値はＤＳＰによって設定されます。飽和検出ロジックでは、生データの絶対値を使っているのでこれらの値は正符号の値となります。デフォルト状態では、near_sat_value は A/D 変換器のフルスケールの約８０％(26214)に設定され、sat_value は A/D 変換器のフルスケールに設定されています。

sat_value = 32768-2^(16-ADC bits)

```
int                 near_sat_value;     /* offset 0x00e0 */
int                 sat_value;          /* offset 0x00e1 */
```

RATE_ADDRESS
RATE_DIVISOR
RATE_COUNT

レイト計算をコントロールするためのデータを含んでいます。rate_address は レイト計算に用いるデータのアドレスです。ＤＳＰは、８つの連続する値に対するレイトを計算します。（例　filter3 データのレイトを計算するには、rate_address を 0x00a8 にセットして下さい。）rate_divisor は レイトをどれくらいの頻度で計算するかを示します。rate_divisor が 1 なら、レイトはセンサの全帯域幅で計算されます。rate_divisor が 200 なら、レイトは 200 サンプル毎に計算されます。rate_divisor は 1〜65536 の間の値をとることができます。65536 サンプル毎にレイトを計算するためには、rate_divisor を 0 にセットして下さい。rate_count は 0 でスタートし、その値が rate_divisor に等しくなるまでカウントします。その時点でレイトは計算され、rate_count は 0 にリセットされます。新しいレイトディバイザ(rate divisor) をセットするときに、レイトディバイザよりも 1 小さい値を rate_count にセットするのは良い方法です。こうすると、レイト計算をスタートさせるための時間を最小にできます。

```
int                 rate_address;       /* offset 0x00e2 */
unsigned int        rate_divisor;       /* offset 0x00e3 */
unsigned int        rate_count;         /* offset 0x00e4 */
```

COMMAND_WORD2
COMMAND_WORD1
COMMAND_WORD0

ＤＳＰにコマンドを送るために使われるロケーションです。この使い方はコマンドによって異なりますので、詳しくはコマンドの定義(Command Definitions,p.27) の項を参照して下さい。一般的には、値をいろいろなメモリロケーションに入れてから、command word を command_word0 に入れます。ＤＳＰはコマンドを処理し、エラーがなければ command_word0 に 0 を、エラーが起こった場合には負の数を入れます。コマンドロケーションはその数とは逆になっていますので注意して下さい。（つまり、command_word2 は command_word1 の前に来ます。）

```
int                 command_word2;      /* offset 0x00e5 */
int                 command_word1;      /* offset 0x00e6 */
int                 command_word0;      /* offset 0x00e7 */
```

COUNT1_COUNT6

対応するフィルタが計算される毎にインクリメントされる符号なしカウンタです。Filter1 はセンサデータの帯域幅で計算されます。従ってこのカウンタは、標準センサでは 8KHz でインクリメントされます。残りのカウンタはその前のカウンタの 1/4 の間隔でインクリメントされるので、2KHz, 500Hz, 125Hz ---- でカウントすることになります。

```
unsigned int        count1;             /* offset 0x00e8 */
unsigned int        count2;             /* offset 0x00e9 */
unsigned int        count3;             /* offset 0x00ea */
unsigned int        count4;             /* offset 0x00eb */
unsigned int        count5;             /* offset 0x00ec */
unsigned int        count6;             /* offset 0x00ed */
```

ERROR_COUNT

データ受取エラーのカウントです。このカウンタが急激に変化している場合には、センサケーブルの接続不良またはその他のハードウェアに問題がある可能性があります。たいていの実装状況では、error_count は全く変化しないはずです。しかし、極めて雑音の多い環境では、ハードウェアに問題がない場合でも時々エラーが発生することがあります。この様な環境のもとでも、センサが完全にグラウンドされていれば多くの場合エラーを防ぐことが出来ます。カウンタがエラーを検出した場合には、そのサンプルデータは無視されます。

```
unsigned int        error_count;        /* offset 0x00ee */
```

COUNT_X

ＤＳＰがそのジョブキューをサーチし、実行すべきものがない場合にインクリメントされるカウンタです。ＤＳＰが使用可能なアイドルタイムの総量を示します。またＤＳＰが正常に動作しているかどうかを判断するためにも使用できます。

```
unsigned int        count_x;            /* offset 0x00ef */
```

WARNINGS
ERRORS

各々 warning bits, error bits が書き込まれています。これらの２ワードのフォーマットについては 19 ページで説明します。

```
warning_bits        warnings;           /* offset 0x00f0 */
error_bits          errors;             /* offset 0x00f1 */
```

THRESHOLD_BITS

ロードエンベロープのモニタ結果を出力するワードです。詳細については load_envelopes(p.14) 及び thresh_struct(p.21) を御参照下さい。

```
int                 threshold_bits;     /* offset 0x00f2 */
```

LAST_CRC

実際に計算された CRC を示す値です。CRC は Cydlic Redundancy Code の略称です。これはゼロであるはずです。詳しくは、cal_crc_bad の説明(p.19)を御参照下さい。

```
int                 last_crc;           /* offset 0x00f3 */
```

EEPROM_VER_NO
SOFTWARE_VER_NO

eeprom_ver_no はセンサの EEPROM のバージョンナンバーが書き込まれています。EEPROM のバージョンナンバーは 0 から 255 の間です。
software_ver_no はソフトウェアのバージョンナンバーが書き込まれています。バージョン 3.02 は 302 としてストアされます。

```
int                 eeprom_ver_no;      /* offset 0x00f4 */
int                 software_ver_no;    /* offset 0x00f5 */
```

SOFTWARE_DAY
SOFTWARE_YEAR

ＤＳＰを動作させているソフトウェアのリリース日を示します。day **は日で、うるう年でない年では、１月１日は１、１２月３１日は３６５として表示します。**

```
int                 software_day;       /* offset 0x00f6 */
int                 software_year;      /* offset 0x00f7 */
```

SERIAL_NO
MODEL_NO

センサのシリアル番号及びモデル番号です。

```
unsigned int        serial_no;          /* offset 0x00f8 */
unsigned int        model_no;           /* offset 0x00f9 */
```

CAL_DAY
CAL_YEAR

センサをキャリブレーションした日を示します。day **は 日で、うるう年以外の場合、１月１日は１、１２月３１日は３６５と表示されます。**

```
int                 cal_day;            /* offset 0x00fa */
int                 cal_year;           /* offset 0x00fb */
```

UNITS
BITS
CHANNELS

units は、センサのフルスケールを示すために用いられる工学単位を定義するものです。これらの値の意味は force_units structure(P.20) の項で説明します。

bits は、センサに使用されている A/D 変換器の分解能を示します。

channels は、センサのどのチャンネルが送信可能かを示すビットフィールドです。bit 0 がアクティブであれば、センサは channel 0 を送信でき、bit13 がアクティブであれば、channel 13 を送信できるということです。このビットは、センサがその対応するチャンネルを送信していなくてもアクティブになり得ます。あるセンサでは どのチャンネルを送るかを設定でき、このビットフィールドは 送信可能なチャンネルの情報を含むだけで、現在のコンフィギュレイションに関する情報は含んでいません。どのチャンネルが現在送信されているかを見るためには、raw_channel array (p.5) の中の raw_time fields (p.16) をモニタして下さい。時間が周期的に変化しておれば、そのチャンネルは受信しているということになります。

```
force_units          units;              /* offset 0x00fc */
int                  bit;                /* offset 0x00fd */
int                  channels;           /* offset 0x00fe */
```

THICKNESS

フランジからフランジまでのセンサの全厚みを示します。厚みの工学単位は units(p.14)に書き込まれた単位です。この値を用いると、センサ中心からフランジ面への座標変換が容易に行えます。

```
int                  thickness;          /* offset 0x00ff */
```

LOAD_ENVELOPES

ロードエンベロープを記述するためのテーブルです。このテーブルには１６個のロードエンベロープスロットの設定が可能です。各スロットは１６ワードの大きさがあり、０－１５の番号がつけられています。各々のロードエンベロープはスロットの始めから書き始める必要がありますが、そのスロットを最後まで満たす必要はありません。また、ひとつのロードエンベロープをひとつのスロットよりも大きくする事も可能です。本ソフトウェアでは５０の閾値をアクティブにした状態で満足に動作することが確認されています。この様な大きさのロードエンベロープは１６のスロットのうちの４つを占有します。ロードエンベロープのデータは、ＤＳＰにとって最も効率のよい配列でレイアウトされます。構造の詳細については le_struct structure の定義の項(p.21) を参照してください。

```
le_struct[0x10]      load_envelopes;     /* offset 0x0100 */
```

TRANSFORMS

座標変換の情報を記述するためのテーブルです。このテーブルには１６個の座標変換スロットの設定が可能です。各スロットは１６ワードの大きさがあり、０－１５の番号がつけられています。各々の座標変換はスロットの始めから書き始める必要がありますが、そのスロットを最後まで満たす必要はありません。また、ひとつの座標変換はひとつのスロットよりも大きくすることができます。ひとつの座標変換の長さは２*（リンク）＋１ワードとなります。したがって、ひとつのスロットは７リンクの座標変換を含むことができ、２スロットでは１５リンクの座標変換を含むことができます。配列の詳細については、transform structure の定義の項(p.24) を参照してください。

```
    transform[0x10]     transforms;          /* offset 0x0200 */

};
```

データストラクチャの詳細

RAW_CHANNEL

raw data（生データ）は、ＤＳＰにより容易に迅速な処理ができるようなフォーマットでストアされます。raw_channel structure は、単一のデータのフォーマットを示しています。各チャンネルは４ワード占有します。

RAW_TIME

raw_time は、各チャンネル毎のデータが受信された時点での、ＤＳＰ内部クロックによる時間を示す数字です。クロックは、ＤＳＰのサイクルタイムの 1/10 で動いています。ＤＳＰは最も遅いもので 10Mhz で動作しています。したがって、10Mhz のＤＳＰでは raw_time のクロックは 1Mhz です。

RAW_DATA

raw_data は、センサから受け取ったままのデータです。ここには１６チャンネルのデータが表示されています。チャンネル０は、センサにおける励起電圧を示しています。これは、ケーブルの長さなどの影響による電圧の降下を補正するために使用します。チャンネル１～６にはクロストークを含んだＦｘからＭｚのデータが表示されます。チャンネル７にはセンサのキャリブレーションデータが表示されます。チャンネル８～１５は特殊品のセンサのみに使用されます。

```
struct  raw_channel
{
     unsigned int    raw_time;
     int             raw_data;
     int[2]          reserved;
};
```

FORCE_ARRAY

force_array **ストラクチャは、干渉を除去しフィルタリングした後の力データのレイアウトを示すものです。**

```
struct  force_array
{
     int            fx;
     int            fy;
     int            fz;
     int            mx;
     int            my;
     int            mz;
     int            v1;
     int            v2;
};
```

SIX_AXIS_ARRAY

six_axis_array **ストラクチャは、オフセットとフルスケールのレイアウトを示しています。**

```
struct   six_axis_array
{
     int            fx;
     int            fy;
     int            fz;
     int            mx;
     int            my;
     int            mz;
};
```

VECT_BITS

vect_bits ストラクチャは､ベクトル計算にどの軸を用いるのかを示すためのレイアウトです。各ビットは、ひとつの軸の選択することを意味します。fx bit は、１６進数の 0x0001 に対応し、mz bit は 0x0020 に対応します。たとえば fx,fy,mx,my という軸を使用するには、0x002b ということになります。ベクトル１のデフォルトは力ベクトルであり、ベクトル２のデフォルトはモーメントベクトルです。このデフォルトを変更することも可能です。V1 bit､あるいは V2 bit を変更すると、各ベクトルはデフォルトの逆になります。つまり、V1 に 1 をセットすることによって両方をモーメントベクトルにすることが出来ます。

```
struct  vect_bits
{
     unsigned       fx : 1;
     unsigned       fy : 1;
     unsigned       fz : 1;
     unsigned       mx : 1;
     unsigned       my : 1;
     unsigned       mz : 1;
     unsigned       changeV1 : 1;
     unsigned       changeV2 : 1;
     unsigned       reserved : 8;
};
```

WARNING_BITS

warning_bits ストラクチャは、警告を与えるワードでありビットパターンで表示します。bit0(lsb)から bit15(msb)を用いて表示します。

XX_NEAR_SAT

xx_near_sat bits は、そこに示された軸が飽和値近辺にまで到達したか、あるいはそれを越えたことを示しています。

```
struct  warning_bits
{
     unsigned       fx_near_sat : 1;
     unsigned       fy_near_sat : 1;
     unsigned       fz_near_sat : 1;
     unsigned       mx_near_sat : 1;
     unsigned       my_near_sat : 1;
     unsigned       mz_near_sat : 1;
     unsigned       reserved : 10;
};
```

ERROR_BITS
XX_SAT
MEMORY_ERROR
SENSOR_CHANGE

error_ bits ストラクチャは、エラーを示すワードであり、ビットパターンで表示します。bit0(lsb)から bit15(msb)を用いて表示します。xx_sat bits は、そこに示された軸が飽和値に至った、あるいはそれを越えたということを示しています。memory_error bit は、電源投入時にボード上のＲＡＭに問題が検出されたことを示します。sensor_change bit は、先に接続されていたものとは別のセンサがＣＲＣチェックをパスしたということを示しています。このビットはラッチされますので、リセットして下さい。

SYSTEM_BUSY

system_busy bit は、ＤＳＰがＢＵＳＹ状態にあり、力データを計算できない状態にあることを示します。新しい座標変換やフルスケールが変更されると、このようなことが起こります。安定するまでには約４ｍｓを必要としますので、高速で力フィードバック制御をおこなっている場合にはご注意下さい。新しいセンサが接続された場合にもシステムがＣＲＣを再計算するためにこのビットがアクティブになります。

CAL_CRC_BAD

cal_crc_bad bit は、ＣＲＣ計算の結果が０にならなかったことを示します。
ＣＲＣとは Cyclic Redundancy Code （巡回冗長符号）の略称で、データコミュニケーションにおいてデータの一致性を決定する方法です。本システムでは力データとともにセンサ中のＲＯＭに記録されているキャリブレーションデータがＤＳＰに送られています。キャリブレーションデータの最後にはＣＲＣが付加されており、センサから受け取ったキャリブレーションデータの一致性を決定することを支援します。ＣＲＣが０にならない場合は二つの原因が考えられます。ひとつめは、全てのキャリブレーションデータを受信していないとき、ふたつめは、キャリブレーションデータそのものが正しくなかったときです。通常、センサは１秒間に３０セット以上のキャリブレーションデータを送信しています。従って、センサを起動して数秒経過してもＣＲＣが０にならない場合にはセンサのキャリブレーションデータに問題があるということです。

WATCH_DOG
WATCH_DOG2

watch_dog **及び** watch_dog2 bits **は、プロセッサではなくセンサのウォッチドッグです。** watch_dog **はセンサケーブル中のデータラインを監視するものであり、**watch_dog2 **はデータラインとクロックラインを監視するものです。**watch_dog **が異常を示さなくとも** watch_dog2 **が異常を示す可能性もあります。この場合はデータラインは正常で、クロックラインに異常が生じていることを示しています。両方の** watch dog **が異常を示した場合にはデータは正常に受信できていないことになります。**

```
struct   error_bits
{
     unsigned       fx_sat : 1;
     unsigned       fy_sat : 1;
     unsigned       fz_sat : 1;
     unsigned       mx_sat : 1;
     unsigned       my_sat : 1;
     unsigned       mz_sat : 1;
     unsigned       reserved : 4;
     unsigned       memory_error : 1;
     unsigned       sensor_charge : 1;
     unsigned       system_busy : 1:
     unsigned       cal_crc_bad : 1;
     unsigned       watch_dog2 : 1;
     unsigned       watch_dog : 1;
};
```

FORCE_UNITS

force_units **には、センサに使用している単位を列挙しています。（１台のセンサに使用されているのはこの中のどれかひとつです。）**

0 - lbs_in-lbs_mils: pound, inch*pound, inch*1000
1 - N_Nm*10_mm*10: Newton, Newton*meter*10, mm*10
2 - kgf*10_kgfcm_mm*10: kilogram-force*10, kilogram-Force*cm, mm*10
3 - klbs_kin-lbs_mils: kilopound, kiloinch*pound, inch*1000

```
enum force_units
{
     lbs_in-lbs_mils,
     N_dNm_mm10,
     kgf*10_kgFcm_mm10
     klbs_kin-lbs_mils
     reserved_units_4
     reserved_units_5
     reserved_units_6
     reserved_units_7
};
```

THRESH_STRUCT

thresh_struct は、ロードエンベロープ内の一組の閾値のレイアウトを示すストラクチャです。各ロードエンベロープにはいくつかのストラクチャを含むことが出来ます。

DATA_ADDRESS

data_address は、その閾値のためのデータアドレスです。閾値はどのデータにも設定することが出来ます。フィルタをかけた力データやフィルタをかけていない力データをモニタすることも出来ると同時に、生データ、レイトデータ、カウンタ、あるいはエラーや警告ワードをモニタすることも可能です。

THRESHOLD

threshold は、データがそれ上まわったあるいは下まわった場合に、ビットをセットする閾値です。すなわち、ＧＥ（greater than equal）で閾値を設定した場合には、データが threshold を上まわったときに、ＬＥ（less than equal）で閾値を設定した場合には、データが threshold を下まわったときに bit_pattern に指定したビットが 0x00f2 の threshold 変数にＯＲ加算されます。

BIT_PATTERN

bit_pattern は、閾値を越えた場合にセットされるビットです。

```
struct  thresh_struct
{
     int            data_address;
     int            threshold;
     int            bit_pattern;
};
```

LE_STRUCT

le_struct は一組のロードエンベロープのレイアウトを示すストラクチャです。ここでは４点の閾値を設定していますが、実際にはさらに多くの閾値を設定することが出来ます。ただし、４点以上の閾値を設定した場合にはそれに続くロードエンベロープの領域と重なりますので、それを理解した上で引き続くロードエンベロープは使用しないで下さい。また、閾値は初めにＧＥタイプ、その後にＬＥタイプをまとめて設定しなければなりません。ＧＥが上限設定、ＬＥが下限設定とお考え下さい。

LATCH_BITS

latch_bits は、ラッチする必要のあるビットを示すビットパターンです。ここにセットされたビットはデータが一度閾値を越えるとラッチされ、データが戻ってもリセットされません。

リセットは reset_bit command で行う必要があります。

NUMBER_OF_GE_THRESHOLDS
NUMBER_OF_LE_THRESHOLDS

これらは、ＧＥ threshold と ＬＥ threshold の数を示すものです。

```
struct  le_struct
{
     int         latch_bits
     int         number_of_ge_thresholds;
     int         number_of_le_thresholds;
     thresh_struct[4]      thresholds;
     int         reserved;
};
```

アドレス	データ	
0	latch Pattern	ラッチするビットのパターン
1	No of GE	ＧＥ閾値の数
2	No of LE	ＬＥ閾値の数
3	data addr	最初のＧＥのアドレス
4	threshold	最初のＧＥの閾値
5	bit pattern	最初のＧＥのビットパターン
6	data addr	２番目のＧＥのアドレス
7	threshold	２番目のＧＥの閾値
8	bit pattern	２番目のＧＥのビットパターン
	•	
	•	
	•	
GE*3	data addr	最後のＧＥのアドレス
GE*3+1	threshold	最後のＧＥの閾値
GE*3+2	bit pattern	最後のＧＥのビットパターン
GE*3+3	data addr	最初のＬＥのアドレス
GE*3+4	threshold	最初のＬＥの閾値
GE*3+5	bit pattern	最初のＬＥのビットパターン
	•	
	•	
	•	
(GE+LE)*3	data addr	最後のＬＥのアドレス
(GE+LE)*3+1	threshold	最後のＬＥの閾値
(GE+LE)*3+2	bit pattern	最後のＬＥのビットパターン

図４　ロードエンベロープストラクチャ

LINK_TYPES

link_types は、座標変換のための変数を列挙したものです。end transform packet は transform chain の最後に付加するものです。translate および rotate はそれぞれ、平行移動、回転移動に使用するためのものです。negate all axes は全ての軸の正負を逆転します。これは座標系をデフォルトの左手系から右手系に変更する場合に使用します。

0 - end tramsform packet
1 - translate along X axis (TX)
2 - translate along Y axis (TY)
3 - translate along Z axis (TZ)
4 - rotate about X axis (RX)
5 - rotate about Y axis (RY)
6 - rotate about Z axis (RZ)
7 - negate all axes

```
enum  link_types
{
      end_x_form,
      tx,
      ty,
      tz,
      rx,
      ry,
      rz,
      neg
};
```

TRANSFORM

transform は、座標変換の内容を示すストラクチャです。変換の順序は任意であり、指定したとおりの順序で行われます。各リンクは２ワードより成り、初めのワードはタイプを示し、後のワードは移動量(link_amount)を示します。タイプは LINK_TYPES(P.23) に示したとおりです。平行移動量は UNITS(P.14) に示した長さの単位で設定し、回転移動量は次のように設定します。

amount = degrees*32768/180　　or　　amount = radians*32768/π

回転移動の場合は１６ビットの整数で０～３６０度をカバーできますが、平行移動の場合には移動できる範囲に限度があります。一つのリンクで移動できない距離であっても、二つ以上のリンクを順序よく組み合わせることにより移動させることが出来る場合があります。

```
struct  links
{
      link_types      link_type
      int             link_amount;
};

struct  transform
{
      links[8]        link;
};
```

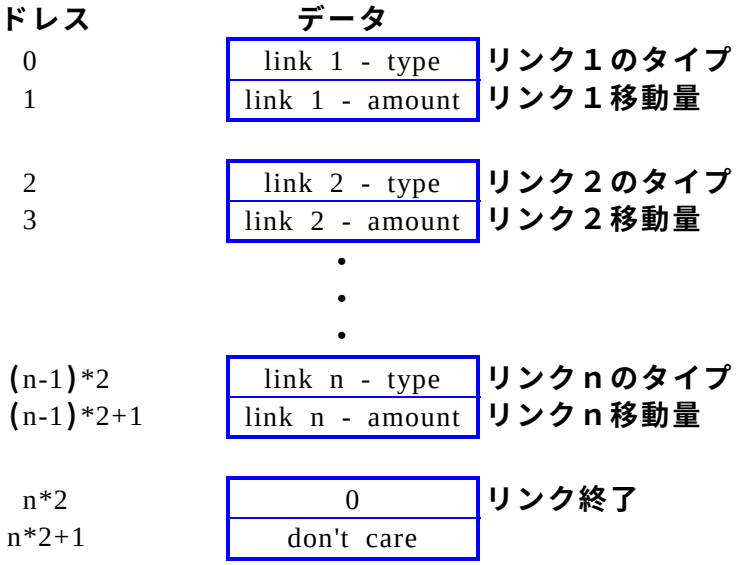

図５　座標変換ストラクチャ

Ｃ言語入門

「データロケーションと定義」及び「データストラクチャ定義」の項で説明したデータの宣言はＣ言語形式で示しています。Ｃ言語を用いたのは、それが現在最も広く使われているコンピュータ言語であるからです。しかし、Ｃ言語をこれまであまり取り扱ったことがない方々のために、簡単にＣ言語のデータ宣言について説明します。

/*と*/はコメント文の初めと終わりを示しています。セミコロン(;)はステートメントの最後を示し、コンマ(,)はリスト中のアイテムを分けるために使われます。１６進数は、数字の最初に 0x を付けて表します。16 を１６進数で表すと、0x0010 となり、157 は 0x009d となります。変数を宣言するときには、変数の型を最初に示し、その次に変数名を示します。sat_value という変数名の整数型変数を宣言する場合には、次のようになります。

```
int             sat_value;
```

int はＣ言語では符号付整数を示します。整数のサイズは、Ｃコンパイラが書かれた機械によって変わります。本システムで使用しているＡＤＳＰ－２１０５は１６ビットの情報を一度に処理するので、整数のサイズは１６ビットです。したがって、整数は-32768～+32767 までの値をとることができます。符号なし整数型でも整数型とサイズは同じですが、符号がありません。したがって、次の

```
unsigned int    countx;
```

は、countx という名の変数を宣言することになり、0～65535 までの値をとることができます。

Ｃ言語には、データをひとかたまりとしてストラクチャ（構造体）に結合する機能があります。struct の宣言は、データストラクチャを定義します。struct を定義した後、struct の型によって変数が宣言され、struct の個々のフィールドをアクセスできるようになります。force array structure を宣言するには、次のようになります。

```
struct     force_structure
{
      int        fx;
      int        fy;
      int        fz;
};
```

ここでは、fx,fy,fz という名の３つの整数フィールドを有する force_structure という名のストラクチャを宣言しています。これらの３つのフィールドは fx が最下位または最初のアドレス、fz が最上位または最後のアドレスになるようにメモリ内に配列します。ストラクチャを使って変数を定義するには、ストラクチャ名は変数宣言における変数の型として使用されます。

```
force_structure filter0;
```

は、force_structure の型の filter0 という名の変数を宣言しています。前の force_structure の宣言を用いると、fx,fy,fz という名の３つのフィールドを持つことになります。

Ｃ言語によるデータの配列は、通常のデータ宣言における変数の型に[n]をつけることにより

宣言されます。[n]の中の n は配列に含まれるエレメントの数を示します。
したがって、

```
force_structure[0x0010]   force_data;
```

は、force_structure の型の 16（１６進数では 0x0010）個のエレメントを有する force_data と呼ばれる変数であることを示します。このとき、force_data は 48(16*3)個の整数から構成されることになります。これは、メモリ上に連続して配列された 16 セットの別々の force data と考えることもできます。

最後に、Ｃ言語には、変数の中の個々のビットやビットのグループを取り扱いやすいようにビットフィールドと呼ばれるデータストラクチャの型があります。ビットフィールドは、ストラクチャの定義の中でエレメント名に:n を付加することによって示されます。:n の中の n はエレメントの使用するビット数を示します。ビットの順序はコンパイラにより指定でき、本システムでは lsb から msb の方向に宣言するものと定義します。これを２進数に換算する場合には最初のビットフィールドが最小の重みを持ち、最後のものが最大の重みを持つことになります。

```
struct  test_bits
{
      fx_bit:1;
      mx_bit:1;
      reserved:14;
}
```

ここでは lsb は fx_bit、２番目のビットは mx_bit、上位の１４ビットは reserved であるストラクチャを宣言していることになります。

コマンドの詳細

コマンド例（０）

引数：

Data_1	ここには command_word_0 をセットするに先立って変更すべきデータの名称を記述しています。
Command_Word_0	0x0000

戻り値：

Data_2	ここにはコマンドの実行によって変化するデータの名称を記述しています。
Command_Word_0	0

（正常にコマンドが実行されると Command_Word_0 が０になります。）

実行時間：

４０μＳ　（平均）
１２５μＳ　（最大）

備考：

このコマンド例では、この章における他のコマンドの表示形式を示しています。
引数と戻り値に関しては、左側に変数の名称を示し右側にそこに設定すべき変数の値、あるいは送り返される変数の値を示しています。
実際にコマンドを実行するときには、まず Data_1 を必要な値に書き換え、次に Command_Word_0 にコマンド番号を書き込みます。ここではコマンド番号は 0x0000 です。その後 Command_Word_0 をモニタし０になるのを待ちます。その時点でコマンドの実行が完了していますので Data_2 の戻り値を読み出して下さい。

メモリリード（１）

引数：

Command_Word_1	**読み出すアドレス**
Command_Word_0	0x0100

戻り値：

Command_Word_2	**読み出したデータ**
Command_Word_0	0

実行時間：

４０μＳ　（平均）
１２５μＳ　（最大）

備考：

メモリは直接読み出すことが出来るので、通常このコマンドを使用することはありません。試験用途でＤＳＰの内部メモリを読み出すときにこのコマンドを使用します。

メモリライト（２）

引数：

Command_Word_2	**書き込むデータ**
Command_Word_1	**書き込むアドレス**
Command_Word_0	0x0200

戻り値：

Command_Word_2	**アドレスより読み出したデータ**
Command_Word_0	0

実行時間：

４０μＳ　（平均）
１２５μＳ　（最大）

備考：

これはメモリへデータを書き込むためのコマンドです。このコマンドはユーザが書き込みを行うと同時にＤＳＰがそのデータを変更する可能性があるような場合に使用します。warning, error, threshold **等のラッチされるビットがそれに相当します。また、このコマンドはデータが書き込まれる前のデータを戻り値として返します。**

ビットセット（３）

引数：

Command_Word_2	**セットすべきビットマップ**
Command_Word_1	**ビットをセットするアドレス**
Command_Word_0	0x0300

戻り値：

Command_Word_2	**アドレスより読み出したデータ**
Command_Word_0	0

実行時間：

４０μＳ　（平均）
１２５μＳ　（最大）

備考：

これは指定したアドレスのデータワードの単一あるいは複数のビットをセットするためのコマンドです。このコマンドはユーザが書き込みを行うと同時にＤＳＰがそのデータを変更する可能性があるような場合に使用します。warning, error, threshold **等のラッチされるビットがそれに相当します。また、このコマンドはデータが書き込まれる前のデータを戻り値として返します。**

ビットリセット（４）

引数：

Command_Word_2	**リセットすべきビットマップ**
Command_Word_1	**リセットするアドレス**
Command_Word_0	0x0400

戻り値：

Command_Word_2	**アドレスより読み出したデータ**
Command_Word_0	0

実行時間：

４０μＳ　（平均）
１２５μＳ　（最大）

備考：

これは指定したアドレスのデータワードの単一あるいは複数のビットをリセットするためのコマンドです。このコマンドはユーザが書き込みを行うと同時にＤＳＰがそのデータを変更する可能性があるような場合に使用します。warning, error, threshold **等のラッチされるビットがそれに相当します。また、このコマンドはデータが書き込まれる前のデータを戻り値として返します。**

座標変換番号（５）

引数：

transforms(@0x200-)	**必要なスロットに座標変換テーブルをセット**
Command_Word_0	0x050?

ここで "?" **は座標変換スロットの番号（**0x0 **～** 0xf**）**

戻り値：

Command_Word_0	0
Transform_Num(@0x77)	**新たに使用を開始した座標の番号**

実行時間：

３～１０ｍＳ（平均）
この実行時間は座標変換のリンク数に依存します。
（２．２５＋０．７５＊リンク数）ｍＳ

備考：

このコマンドを実行する前に座標変換テーブルを作成し書き込んでおいて下さい。座標変換を１種類しか使用しない場合にはアドレス 0x200 **からのスロット０に座標変換テーブルを書き込み、** Command_Word_0 **に** 0x0500 **を書き込んで下さい。このコマンドの実行には数ｍＳを要します。コマンドの実行中には** error_bits **の** system_busy **ビットがセットされます。また、他のコマンドと同様にコマンドの実行が完了すると** Command_Word_0 **が０にリセットされます。**

オフセット番号（６）

引数：

Command_Word_0　　0x060?

ここで "?" **はオフセット番号（**0x0 **～** 0xf**）**

戻り値：

Command_Word_0　　0

offset_num(@0x8e)　　**新たに使用を開始したオフセット番号**

offset(@0x88-)　　**指定したオフセット値**

実行時間：

１５０μＳ　（平均）

２２５μＳ　（最大）

備考：

本システムでは１６通りのオフセット値をセーブしておくことが出来ます。使用する工具を交換するような場合にこの機能が必要になります。このコマンドは使用するオフセット番号を指定すると同時に、セーブされたオフセット値をロードします。後述のセットオフセットコマンドやリセットオフセットコマンドはここで指定しておいたオフセット番号に対して実行されます。複数のオフセットを使用しない場合にはこのコマンドは必要なく、デフォルトのオフセット番号０が常に使用されることになります。使用中のオフセット値を変更せずにオフセット番号を変更したい場合には、アドレス 0x8e **の** offset_num **に変更したい番号を直接書き込んで下さい。**

セットオフセット（７）

引数：

offset(@0x88-)	**セットしたいオフセット値**
Command_Word_0	0x0700

戻り値：

Command_Word_0	0
current offset table entry	0x88 **にセットされたオフセット値**

実行時間：

１５０μＳ　（平均）
２２５μＳ　（最大）

備考：

アドレス 0x88 **にセットされているオフセット値を直ちに使用すると同時に、使用中のオフセット番号にその値をセーブします。このコマンドを使用しなくても** 0x88 **にセットされたオフセット値はセット後最大２ｍＳ後に使用が開始されますが、このコマンドを実行することによって直ちに使用が開始されます。**

リセットオフセット（８）

引数：

Command_Word_0	0x0800

戻り値：

Command_Word_0	0
offsets(@0x88-)	**力データ出力が０になるようなオフセット値**
current offset table entry	**同上**

実行時間：

１５０μＳ　（平均）
２２５μＳ　（最大）

備考：

アドレス 0xa0 **からの** filter2 **出力が０になるようなオフセット値を採用すると同時に、使用中のオフセット番号にセーブする。**

セットベクトル（９）

引数：

Command_Word_0	0x09??
	ここで "??" **はベクトル計算に使用する軸をビットマップで指定します。**

戻り値：

Command_Word_0	0
vect_axes(@0x8f)	Command_Word_0 **のビットマップがセットされます。**
full_scale(@0x86-)	**選択した軸のフルスケールの最大値になります。**

実行時間：

４０μＳ　（平均）
１２５μＳ　（最大）

備考：

V1 **と** V2 **の２種類のベクトルに使用する軸を指定します。各ビットの意味は** lsb **より、**V1x,V1y,V1z,V2x,V2y,V2z **です。デフォルトでは** V1 **は力、** V2 **はモーメントを表しますが、**0x40 **ビットをセットすることにより、**V2 **も力ベクトルに使用され、**0x80 **ビットをセットすると** V1 **がモーメントベクトルに使用されます。**

例：V1 **を** Fx,Fz**、**V2 **を** Fy,Fz **のベクトルと指定する場合．．．**0x75

V1 **を** Mx,My**、**V2 **を** Mx,Mz **のベクトルと指定する場合．．．**0xab

セットフルスケール（１０）

引数：

full_scale(@0x80-)	**設定したいフルスケール**
Command_Word_0	0x0a00

戻り値：

Command_Word_0	0

実行時間：

３～１０ｍＳ（平均）
この実行時間は座標変換のリンク数に依存します。
（３．４＋０．７５＊リンク数）ｍＳ

備考：

このコマンドの実行には数ｍＳを要します。コマンドの実行中には error_bits **の** system_busy **ビットがセットされます。また、他のコマンドと同様にコマンドの実行が完了すると** Command_Word_0 **が０にリセットされます。**

このコマンドを実行すると座標変換の計算も同時に行われます。従って、座標変換番号や有効になっている座標変換テーブルを変更してある場合にこのコマンドを実行すると座標系が変わる事になります。これは便利な事もありますが、予想外の結果を招く事もありますので、十分に注意して実行して下さい。

適切なフルスケールの設定方法については、minimum full scale(P.6)**の項を参照して下さい。**

リセットピーク（１１）

引数：

peak_address(@0x7f)	**ピークを調べたいデータのアドレス**
Command_Word_0	0x0b00

戻り値：

Command_Word_0	0
minimums(@0xd0-)	**最後にピークをリセットしてからの最小値**
maximums(@0xd8-)	**最後にピークをリセットしてからの最大値**

実行時間：

４０μＳ　（平均）
１２５μＳ　（最大）

備考：

本システムでは任意のデータの最小、最大の各ピーク値を監視出来ます。そのピーク値はこのコマンドによってアドレス 0xd0 **～** 0xdf **に出力してラッチすると同時に内部のピーク値を現在の出力値にリセットします。これらのデータは常に変化しているわけではありませんがこの方法を取る事によって確実に同じデータサイクルのピーク値を測定する事が出来ます。**

リードピーク（１２）

引数：

peak_address(@0x7f)	**ピークを調べたいデータのアドレス**
Command_Word_0	0x0c00

戻り値：

Command_Word_0	0
minimums(@0xd0-)	**最後にピークをリセットしてからの最小値**
maximums(@0xd8-)	**最後にピークをリセットしてからの最大値**

実行時間：

４０μＳ　（平均）
１２５μＳ　（最大）

備考：

本システムでは任意のデータの最小、最大の各ピーク値を監視出来ます。そのピーク値はこのコマンドによってアドレス 0xd0 ～ 0xdf **に出力してラッチします。これらのデータは次にコマンドを実行するまで変化しません。この方法を取る事によって確実に同じデータサイクルのピーク値を測定する事が出来ます。**

コマンドの使用例

ここでは、コマンドの使用例を疑似プログラムで示しています。１行の中で＃＃以降はコメントです。いくつかの関数をあらかじめ定義します。

```
readData (addr)
```

これは addr で示したＤＳＰ上のアドレスの 16bit データを読み出す関数です。

```
writeData (addr, data)
```

これは addr で示したＤＳＰ上のアドレスに data を書き込む関数です。

```
write _____________
```

これは下線で示した部分をディスプレイに表示する関数です。

例１－ソフトウェアバージョンの表示

使用中のＤＳＰのソフトウェアバージョンを表示する例です。バージョン No.はアドレス 0xf5 にあります。

```
write'ｿﾌﾄｳｪｱﾊﾞｰｼﾞｮﾝ'
write(readData (0xf5) / 100)
```

例２－計算後のＦｘ力データの表示

この例ではアドレス 0x80 のＦｘフルスケールを読み込み、アドレス 0xa0 のフィルタ２の力データを換算しています。

```
write'Fx ﾌｨﾙﾀ 2'
write(readData (0xa0) / 16384 * readData (0x80))
```

例３－オフセットのリセット

全ての軸の力データをゼロにするためのリセットオフセットのコマンド（p.34）を実行する例です。風袋除去に使用します。アドレス 0xe7 の command_word0 に 0x0800 を書き込むことによってコマンド（８）が実行されます。

```
writeData (0x00e7, 0x0800)
```

例４－Ｆｚのオフセット変更

ここではＦｚの出力を２０にするようなオフセットを設定する例を示します。アドレス 0xa2 **のフィルタ２データとアドレス** 0x8a **の現時点のオフセット値から風袋除去前のＦｚデータが計算できます。その値から２０引いたデータをオフセットとして設定することによって、Ｆｚの出力は２０になります。**

```
writeData (0x008a, readData (0x00a2) + readData (0x008a) - 20)
```

上記の１行だけでオフセットを変更した場合、オフセットが変更されたことをＤＳＰが認識するまで待たなければなりません。これには最大２ｍＳかかりますがコマンド（７）を実行することにより、即座にオフセットを変更することが出来ます。

```
## Ｆｚを２０に設定
writeData (0x008a, readData (0x00a2) + readData (0x008a) - 20)
## 他の軸も必要に応じ設定


## 最後にセットオフセットコマンドを実行
writeData (0x00e7, 0x0700)
```

例５－ベクトルの設定

これはベクトル計算に使用する軸を設定する例です。デフォルトにではＶ１とＶ２のベクトルは、それぞれ力とモーメントのベクトルであり、ｘ，ｙ，ｚ軸を全て使っています。Ｖ１とＶ２に用いる軸を変更するにはコマンド（９）(p.35)を使用します。次の例は、Ｖ１にＦｘとＦｙを用い、Ｖ２にＦｘ、Ｆｙ、Ｆｚを用いるように設定を行っています。アドレス 0xe7 **の** command_word0 **に** 0x097b **を書き込むことによって実行されます。**

```
writeData (0x00e7, 0x097b)
```

ここで書き込んだビットマップデータ 0x7b **は次のようにして決まります。**

lsb	**Ｖ１ｘ**	**1**
	Ｖ１ｙ	**1**
	Ｖ１ｚ	**0**
	Ｖ２ｘ	**1**
	Ｖ２ｙ	**1**
	Ｖ２ｚ	**1**
	Ｖ２変更	**1**
msb	**Ｖ１変更**	**0**

例６－座標変換

ここでは単純な座標変換の例を示します。この座標変換はＹ軸まわりに１８０度回転させるだけのものです。座標変換テーブル０を用います。まずアドレス 0x0200 **の座標変換テーブルにＹ軸まわりの回転を意味する５を書き込みます。次に回転量１８０度を書き込みます。この値は(**p.24**)で述べたとおり－３２７６８と換算しています。最後に末尾を示す０を書き込みます。アドレス** 0xe7 **の** command_word0 **に** 0x0500 **を書き込むことによってテーブル０でコマンド（５）が実行されます。**

```
writeData (0x0200, 5)                    ## Ｙ軸まわりの回転
writeData (0x0201, -32768)               ## －１８０度回転
writeData (0x0202, 0)                    ## 座標変換０はここまで
writeData (0x00e7, 0x0500)               ## 座標変換０を使用
```

例７－複雑な座標変換

ここでは複雑な座標変換の例を示します。この座標変換ではまずＺ軸まわりに４５度回転させます。座標変換テーブル２を用います。アドレス 0x0220 **の座標変換テーブルにＺ軸まわりの回転を意味する６を書き込みます。次に回転量４５度を書き込みます。この値は(**p.24**)で述べたとおり８１９２と換算しています。２番目のリンクとしてセンサの厚さの半分だけＺ軸に沿って並行移動します。こうすることによってセンサのフランジ面に座標の原点が移動します。最後に末尾を示す０を書き込みます。アドレス** 0xe7 **の** command_word0 **に** 0x0502 **を書き込むことによってテーブル２でコマンド（５）が実行されます。**

```
writeData (0x0220, 6)                    ##Ｚ軸まわりの回転
writeData (0x0221, 8192)                 ##４５度回転
writeData (0x0222, 3)                    ##Ｚ軸並行移動
writeData (0x0223, readData (000ff) / 2) ##移動量はセンサ厚の半分
writeData (0x0224, 0)                    ##座標変換２はここまで
writeData (0x00e7, 0x0502)               ##座標変換２を使用
```

例８－ロードエンベロープ

ここでは、複数の閾値を持つロードエンベロープの設定と使い方の例を示します。２つの軸に５種類の閾値を設定します。ＦｘとＦｙの正負両側にフルスケールの１／４の閾値を設定し、各閾値にそれぞれ２つのビットを割り当てて、１つのビットをラッチビットとします。また、力ベクトルにフルスケールの１／２の閾値を設定します。最後に閾値ビット読み取った後それらをリセットします。

```
## ロードエンベロープ２に所望の値を入れる
writeData (0x0120, 0xff00)          ## 上位８ビットはラッチする
writeData (0x0121, 3)               ## GE 閾値（上限値）を３点
writeData (0x0122, 2)               ## LE 閾値（下限値）を２点

## 最初の上限値
writeData (0x0123, 9x0090)          ## フィルタ０、Ｆｘのアドレス
writeData (0x0l24, 4096)            ## フルスケールの 1/4
writeData (0x0125, Ox0101)          ## ビット０と８を使用

## ２番目の上限値
writeData (0x0126, 0x0091)          ## フィルタ０、Ｆｙのアドレス
writeData (0x0127, 4096)            ## フルスケールの 1/4
writeData (0x0128, 0x0202)          ## ビット１と９を使用

## 最後の上限値
writeDeta (0x0129, 0x0096)          ## フィルタ０、ｖ１のアドレス
writeData (0x012a, 8192)            ## フルスケールの 1/2
writeData (0x012b, 0x1010)          ## ビット４と１２を使用

## 最初の下限値
writeData (0x012c, 0x0090)          ## フィルタ０、Ｆｘのアドレス
writeData (0x012d, -4096)           ## フルスケールの-1/4
writeData (0x012e, 0x0404)          ## ビット２と１０を使用

## この閾値からスロット３の領域に入ります
## 最後の下限値
writeData (0x012f, 0x0091)          ## フィルタ０、Ｆｙのアドレス
writeData (0x0130, -4096)           ## フルスケールの-1/4
writeData (0x0131, 0x0808)          ## ビット３と１１を使用

writeData (0x006f, 2)               ## ロードエンベロープ２を使用

write '現在の閾値状況'
write (readData (0x00f2))

## ラッチされたビットのリセット
writeData (0x00e5, 0xff00)          ## 上位８ビットをリセット
writeData (0x00e6, 0x00f2)          ## リセットするアドレス
writeData (0x00e7, 0x0400)          ## コマンド４（ﾋﾞｯﾄﾘｾｯﾄ）を実行
```

データロケーション

網掛け部はデータ書き込み可能

	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
0x00	ch0time	ch0data			ch1time	ch1data		
0x08	ch2time	ch2data			ch3time	ch3data		
〜	〜	〜			〜	〜		
0x38	chEtime	chEdata			chFtime	chFdata		
0x40	'C '	'o '	'p '	'y '	'r '	'i '	'g '	'h '
0x48	't '	' '	'J '	'R '	'3 '	', '	' '	'l '
0x50	'n '	'c '	' '	'1 '	'9 '	'9 '	'3 '	null
0x58								
0x60	shunt fx	shunt fy	shunt fz	shunt mx	shunt my	shunt mz		
0x68	def fs fx	def fs fy	def fs fz	def fs mx	def fs my	def fs mz		load env #
0x70	min fs fx	min fs fy	min fs fz	min fs mx	min fs my	min fs mz		xForm #
0x78	max fs fx	max fs fy	max fs fz	max fs mx	max fs my	max fs mz		peak addr
0x80	fs fx	fs fy	fs fz	fs mx	fs my	fs mz	fs v1	fs v2
0x88	ofs fx	ofs fy	ofs fz	ofs mx	ofs my	ofs mz	ofs #	vect axes
0x90	f0 fx	f0 fy	f0 fz	f0 mx	f0 my	f0 mz	f0 v1	f0 v2
0x98	f1 fx	f1 fy	f1 fz	f1 mx	f1 my	f1 mz	f1 v1	f1 v2
0xa0	f2 fx	f2 fy	f2 fz	f2 mx	f2 my	f2 mz	f2 v1	f2 v2
0xa8	f3 fx	f3 fy	f3 fz	f3 mx	f3 my	f3 mz	f3 v1	f3 v2
0xb0	f4 fx	f4 fy	f4 fz	f4 mx	f4 my	f4 mz	f4 v1	f4 v2
0xb8	f5 fx	f5 fy	f5 fz	f5 mx	f5 my	f5 mz	f5 v1	f5 v2
0xc0	f6 fx	f6 fy	f6 fz	f6 mx	f6 my	f6 mz	f6 v1	f6 v2
0xc8	rate fx	rate fy	rate fz	rate mx	rate my	rate mz	rate v1	rate v2
0xd0	min fx	min fy	min fz	min mx	min my	min mz	min v1	min v2
0xd8	max fx	max fy	max fz	max mx	max my	max mz	max v1	max v2
0xe0	near sat	sat	rate addr	rate div	rate count	comm 2	comm 1	comm 0
0xe8	count 1	count 2	count 3	count 4	count 5	count 6	errors	count x
0xf0	warning	error	threshold	crc	rom ver #	ver no	ver day	ver year
0xf8	serial	model	cal day	cal year	units	bits	chans	thickness

	概要	参照ﾍﾟｰｼﾞ
ch0time,ch0data	ｃｈ０のデータが最後に受信されてからの時間、そのデータの内容	5
shunt fx,..	Ｆｘのシャントの値（ｵﾌﾟｼｮﾝ）	6
def fs fx,...	Ｆｘフルスケールのデフォルト	6
min fs fx,...	Ｆｘフルスケールの最小値	6
max fs fx,...	Ｆｘフルスケールの最大値	7
fs fx,...	Ｆｘのフルスケール、Ｆｘが１６３８４を出力したときの単位付き計算値	7
load env #	現在有効になっているロードエンベロープの番号	6
xForm #	現在有効になっている座標変換の番号	7
peak addr	最大値、最小値を算出するために使用するデータのアドレス	7
ofs fx,...	Ｆｘのオフセット	8
ofs #	現在使用中のオフセットの番号	8
vect axes	合力計算に使用する軸を表すビットマップ	8
f0 fx,...	干渉除去後でフィルタをかけていない力データ	8
f1 fx,...	フィルタ１を通過後の力データ	9
rate fx,...	Ｆｘの差分データ	9
min fx,... , max fx,...	Ｆｘの最小値、最大値	9
near sat, sat	warning word,error word に使用するための飽和点の設定値	10
rate addr	差分データ計算に使用するデータのアドレス	10
rate div	差分データを計算するデータの間隔	10
rate count	差分データの間隔のためのカウンタ	10
comm 2,...	コマンドワード	11
count 1,...	フィルタ１のためのカウンタ	11
errors	エラー回数のカウンタ	11
warning, error	ワーニング、エラーのモニタ	12
threshold	ロードエンベロープのモニタ	12
rom ver #	センサ本体内蔵のＲＯＭのバージョンNo.	12
ver no, ver day	レシーバボードＲＯＭのバージョンNo.と日付	12
serial, model	センサ本体のシリアルNo.とモデルNo.	13
cal day	センサ本体の校正日	13
units	センサに採用されている単位系	14
bits	センサのA/D変換器のｂｉｔ数	14
chans	センサが送信可能なチャンネルのビットマップ	14
thickness	センサの高さ	14

コマンド一覧

コマンド	0上位	0下位	1	2	入力データ	出力データ
ﾒﾓﾘﾘｰﾄﾞ	1	ー	ｱﾄﾞﾚｽ	ー	ー	ﾃﾞｰﾀ(com2)
ﾒﾓﾘﾗｲﾄ	2	ー	ｱﾄﾞﾚｽ	ﾃﾞｰﾀ	ー	ﾃﾞｰﾀ(com2)
ﾋﾞｯﾄｾｯﾄ	3	ー	ｱﾄﾞﾚｽ	ﾋﾞｯﾄﾏｯﾌﾟ	ー	ﾃﾞｰﾀ(com2)
ﾋﾞｯﾄﾘｾｯﾄ	4	ー	ｱﾄﾞﾚｽ	ﾋﾞｯﾄﾏｯﾌﾟ	ー	ﾃﾞｰﾀ(com2)
座標変換	5	座標変換 No.	ー	ー	座標変換 ﾃｰﾌﾞﾙ	座標変換 No.
ｵﾌｾｯﾄ選択	6	ｵﾌｾｯﾄ No.	ー	ー	ー	ｵﾌｾｯﾄNo. 及びｵﾌｾｯﾄ
ｾｯﾄｵﾌｾｯﾄ	7	ー	ー	ー	ｵﾌｾｯﾄ	
ﾘｾｯﾄｵﾌｾｯﾄ	8	ー	ー	ー	ー	ｵﾌｾｯﾄ
ｾｯﾄﾍﾞｸﾄﾙ	9	軸	ー	ー	ー	ﾍﾞｸﾄﾙ軸
ｾｯﾄﾌﾙｽｹｰﾙ	１０	ー	ー	ー	ﾌﾙｽｹｰﾙ	
ﾘｾｯﾄﾋﾟｰｸ	１１	ー	ー	ー	ー	最大、最小
ﾘｰﾄﾞﾋﾟｰｸ	１２	ー	ー	ー	ー	最大、最小

用語の説明

0x0406	１０進数１０３０の１６進表記
ADC	A/D変換器：センサに内蔵された歪ゲージからのアナログ電圧をＤＳＰで処理可能なデジタル信号に変換するためデバイス。
ビットフィールド	個々のビットがそれぞれ異なった意味をもつデータフィールド。
バイト	８ビットデータ。
Fx、Fy、Fz	各々Ｘ方向、Ｙ方向、Ｚ方向の力を示す。
GE	”大なりイコール”の意味。
LE	”小なりイコール”の意味。
lsb	Least Significant Bit：ビット０または２進値０のビットです。
LSB	Least Significant Byte：複数バイトデータで、最小の値を有するバイトを示す。１０進法を例にとると、１２３４のＬＳＢは４。
LSW	Least Significant Word：ＬＳＢ参照。
msb	Most Significant Bit：ｌｓｂ参照。
MSB	Most Significant Byte：ＬＳＢ参照。
MSW	Most Significant Word：ＬＳＢ参照。
Mx、My、Mz	各々Ｘ軸、Ｙ軸、Ｚ軸まわりのモーメントを示す。
ロボット側の視点	本力センサの座標系は、センサのツール側が固定されたとみなし、力はロボット側から加えられるとした時右手の法則にしたがうように配向している。これをロボット側の視点と呼ぶ。
ツール側の視点	力センサのロボット側が固定され、力はセンサのツール側から加えられるとした時の視点。この視点を用いると、本センサは左手系座標となる。これを変更する必要がある時には座標変換の nagate を使用する。
V1、V2	Ｖｅｃｔｏｒ１及びＶｅｃｔｏｒ２の省略形。
ワード	１６ビットデータ。

処理速度について

ＩＦＳレシーバーボードには、非常に高性能なＤＳＰ（デジタル信号プロセッサー）が使用されています。しかし、このチップにも、能力の限界があります。したがって、いくつかのデータ数値を用い計算によって妥協点を見出しています。この章では種々のデータ計算を行ったときの周波数について詳しく述べています。これらの速度に関する詳細はデータを取り扱う上で有用になります。

全ての計算周波数はセンサーのデータ速度に依存します。標準センサーのデータ速度は８ｋＨｚです。ここでは計算周波数をセンサーの速度(bandwidth:帯域幅)によって表します。例えば標準のセンサーにおいて 1/2 bandwidth は４ｋＨｚです。

センサーデータは全帯域幅において干渉除去が行われオフセットが除去されています。フィルタ１とピークデータでも全帯域幅で計算がされています。レイトデータはユーザの設定したパラメータに従ってセンサー速度の１／ｘ（ｘ：１～１６３８４）で計算されます。従ってレイトデータのパラメータに１を設定した場合には全帯域幅において計算されることになります。saturation status bit は全帯域幅でモニタ出来ますが、その他のデータはそれよりも小さい帯域幅で計算されます。

ロードエンベロープの閾値はセンサー速度の１／４で監視されます。フィルタデータはその前段階のフィルタデータの１／４帯域幅で計算されます。

フィルター１ - 1/1 bandwidth
フィルター２ - 1/4 bandwidth
フィルター３ - 1/16 bandwidth
フィルター４ - 1/64 bandwidth
フィルター５ - 1/256 bandwidth
フィルター６ - 1/1024 bandwidth

それぞれの合力はそれぞれのデータから計算しています。合力データ自体にはフィルタをかけていませんがフィルタをかけたデータを用いて合力を計算しています。

フィルタ０の合力 - 1/2 bandwidth
フィルタ１の合力 - 1/4 bandwidth
フィルタ２の合力 - 1/16 bandwidth
フィルタ３の合力 - 1/64 bandwidth
フィルタ４の合力 - 1/256 bandwidth
フィルタ５の合力 - 1/256 bandwidth
フィルタ６の合力 - 1/1024 bandwidth

ユーザーはＤＳＰに対し自由に種々の命令を課すことが出来るので、ＤＳＰに対して重い負荷になる可能性があります。２つの主要因は rate command とロードエンベロープです。２番目に大きな影響を与えるのはホストがＤＳＰからデータを読み出す際にＤＳＰを占拠する時間です。ホストが command_word_0 が 0 に変化することを高速度でモニタしたり、他の用途でＤＳＰのバスを長時間占拠するとＤＳＰの計算速度は遅くなります。正常な使い方をすればこのような原因による速度低下は起こりませんが、非常に早いホストマシンでＤＳＰにアクセスし続けた場合に問題となり得ます。

count_x(p.12)の変化によってＤＳＰの機能している速度を確認することが出来ます。count_xはＤＳＰがジョブキューをサーチしたりすべきことが無いことを確認する毎に増加します。各データパケットにおいて count_x が少なくとも１増加したことを確認することにより、全てのタスクが完了したことを確認できます。従って８ｋＨｚの標準センサの場合１００％の能力が維持されていれば１秒あたりに count_x は８０００以上増加します。

ソフトウェアバージョン 2.0 の場合 count_x の１カウントには通常約５．４μＳの空き時間があります。従って１秒間につき５．４μＳ＊カウント周波数の時間を他の作業に割り当てることが出来ます。一方、rate の処理には一回当たり約３．３μＳ、ロードエンベロープの処理には一回当たり約０．６μＳの時間を要します。

例１
標準の８ｋＨｚセンサにおいてある条件下で count_x は約２００００Ｈｚでカウントされていると仮定します。この場合には（２００００Ｈｚ－８０００Ｈｚ）＊５．４μＳすなわち１秒間に６４．８ｍＳの時間を他の作業に割り当てることが出来ます。これを全てロードエンベロープの処理に割り当てた場合、一つの閾値につき（０．６μＳ＊２０００Ｈｚ）すなわち１秒あたり１．２ｍＳを必要とするので、最大５４（６４．８／１．２）の閾値を設定できることが計算できます。

例２
標準の８ｋＨｚセンサでレイトを１／８００に設定し、ロードエンベロープを使用しなかった場合 count_x の実測値は約３０５００Ｈｚでした。このレイトを１／１に変更した場合のロードエンベロープに設定できる閾値の最大値を計算します。１秒あたりに作業できる時間は（３０５００Ｈｚ－８０００Ｈｚ）＊５．４μＳ＝１２１．５ｍＳです。レイトの設定を変更することによって必要となる作業時間は７９９／８００＊８０００Ｈｚ＊３．３μＳですから１秒あたり２６．４μＳを要することになります。従って残された作業時間は１秒あたりに１２４．５－２６．４＝９５．１μＳと言うことになります。例１でも述べたとおり閾値１点につき１秒あたり１．２ｍＳを必要とするので９５．１／１．２＝７９が設定できる閾値の最大値です。

これらの例は p.14 に述べた５０以上の閾値の使用を推奨するという意味ではありません。しかし個々のアプリケーションに適した条件（特にレイト計算を必要としない場合）においては恐らくこれ以上の閾値を設定できることになります。